National College of Ireland

Distributed Systems

March 2021

Name	Shane Reynolds
Student Number	x21126143
Lecturer	Divyaa Manimaran Elango
Project Domain	Smart Transport System
Project Specification	Smart Railway System

Contents

1 Introduction

For this project, services for a smart transport system will be implemented according to the last digit of student number. It will specifically focus on a railway system that operates in an urban area as well as providing services for intercounty travel. The user can make several interactions with defined services that communicate via client / server gRPC (Google Remote Procedure Call). A remote procedure call is when a computer program pertains the ability to execute commands and code in a separate address space. The gRPC system is Google's adaptation of an RPC that allows for high performance calls between different programming languages in lightweight packages.

The goal of the implementation of this smart railway system is to provide an intuitive, efficient, and user-friendly way for users to interact with the system, and to plan accordingly using the service definitions that are outlined below. For the final project, further services could be implemented such as real time services that allow the user to interact with smart devices on the train.

2 Service 1: Timetable

This service deals with viewing the train timetable for the day, allowing the user to plan ahead. It will return the various train numbers that will help with the booking process in a later service, as well as other information such as the time and price.

2.1 Methods

2.1.1 RPC Method 1

Rpc view (stream stations) returns (stream trainDetails){ } //Returns times, prices, and train number. This is a **bidirectional streaming** RPC.

2.1.2 RPC Method 2

Rpc delays (trainNo) returns (delaysBool) { } //Returns true or false if there are any delays today

2.1.3 RPC Method 3

Rpc amenities (trainNo) returns (stream trainAmenities) { } //returns specific amenities on train

```
Message stations{

        String departStation = 1;

        String arrivalStation = 2;

}
```

```
Message trainDetails{

            String time = 1;

            double price = 2;

            int trainNo = 3;

}

Message trainNo{

            int trainNo = 1;

}

Message delaysBool{

            Boolean delaysBool = 1;

}

Message trainAmenities{

            String catering = 1;

            String bikeSlot = 2;

            String petsAllowed = 3;

}
```

3 Service 2: Booking

This service is used once the user has acquired the train number that they wish to book from the previous timetable service.

3.1 Methods

3.1.1 RPC Method 1

Rpc login (loginRequest) returns (loginReply) { } //Returns success or failure depending on username and password entered. This is for validation of the user.

3.1.2 RPC Method 2

Rpc booking (trainNo) returns (stream bookingConfirmation) { } //Returns a confirmation message of the details of the train the user just booked.

```
Message loginRequest{

        string username = 1;

        string password = 2;

}

Message loginReply{

        string message = 1;

}

Message trainNo{

        int trainNo = 1;

}

Message bookingConfirmation{

        int bookingNo = 1;        //print a random integer

        double price = 2;

        int trainNo = 3;

}
```

4 Service 3: Support

This service deals with various customer support elements that may help with their experience if they have a question. It also contains functionality to provide the user with emergency aid.

4.1 Methods

4.1.1 RPC Method 1

Rpc liveChat (stream chatClient) returns (stream chatServer) { } // Allows the user to interact with a live agent for support. This is a **bidirectional streaming** RPC.

4.1.2 RPC Method 2

Rpc complaint (complaintMsg) returns (confirmation) { } //Complaining returns a ‘complaint sent’ confirmation.

4.1.3 RPC Method 3

Rpc emergency (emergencyReportBool) returns (stream emergencyResponse) { } // The user sends a Boolean (button click), and the server returns multiple strings about emergency contact details.

```
Message chatClient{

        String messageclient = 1; //Overwritten when the user types a new message? Or FAQ.

}

Message chatServer{

        String messageServer = 1; //Overwritten when the agent types a new message? Or FAQ.

}

Message complaintMsg{

        String message = 1;

}

Message confirmation{

        String confMsg = 1;

}
```

```
Message emergencyReportBool{

        Boolean emergency = 1;

}

Message emergencyResponse{

        String safetyDetails = 1;

        Int policeNo = 2;

        Int paramedicNo = 3;

}
```